An Inverse Limit Construction of a Domain of Infinite Lists

Young-il Choo

Computer Science
California Institute of Technology

5204:TR:85

An Inverse Limit Construction of a Domain of Infinite Lists

Young-il Choo

Computer Science Department
California Institute of Technology

5188:TR:85

superceded by
5204:TR:85

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order No. 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

An Inverse Limit Construction of a Domain of Infinite Lists

Young-il Choo
Caltech Computer Science
May 1985, Revised October 1985

Abstract

A domain of infinite lists is constructed by taking the inverse limit of a chain of finite list domains ordered by projection. The resulting space, called L_∞, is shown to be a complete partial order. Its use as a semantic domain for non-terminating programs is illustrated.

1 Introduction

Infinite lists arise when we want to give meaning to non-terminating, yet answer producing, programs. In applicative languages lazy evaluation enables infinite objects to be progressively computed. Precise formulation of infinite lists is necessary for formal reasoning of such programs.

As a start, we might define infinite lists by specifying each element of the list. Unfortunately, this method is inadequate if we want lists whose elements are again infinite lists. Next, consider defining lists by levels. First, we have infinite lists of order one, the elements of which are atoms. Next, the infinite lists of order $n+1$ are obtained by allowing infinite lists of order n or less as elements. Even this construction does not capture all definable infinite lists.

Consider an infinite list whose first element is an infinite list of order one, second element is a list of order two, etc. Clearly, this list is not of any finite order. We shall say a list like this has order ω. The interesting point here is that such lists can actually be generated in a programming language that allows lazy evaluation. Once we have a list of order ω, we can use it to define a list of order $\omega \cdot 2$ and so on (up to ϵ_0).

Infinite objects can be computed only as a limit of finite ones. The notion of a limit presupposes some kind of a topology, or at least an ordering. The ordering we use is that of definedness. By generating a sequence of finite lists that become more and more defined, we can specify an infinite list.

In these notes we explore a method of constructing a domain of infinite lists by taking the inverse limit of the chain of finite lists ordered by projection, and indicate how the meaning of non-terminating programs can be defined. For more detail on the inverse limit construction, good references are Dugundji [66] and Nagata [68].

2 Finite Lists

Given a set of atoms A, make it into a flat lattice by the usual technique of adjoining a bottom element, $\perp_A$, which is less than all the defined elements of A. We indicate the empty list by $\diamond$, and the undefined list by $\perp$.

Definitions

The *finite lists* of length n, L_n, are defined inductively:

$$L_0 = \{\Diamond, \bot\}$$
$$L_1 = (A \cup L_0) \times L_0 \cup L_0$$
$$L_{n+1} = (A \cup L_n) \times L_n \cup L_n.$$

The set of all *finite lists*, F, is the union of all the L_n's. The ordering on F, and therefore for each L_n, is defined as follows:

(1) $\bot \sqsubseteq x$ for all $x \in F$,

(2) $x{:}y \sqsubseteq u{:}v$ if $x \sqsubseteq u$ and $y \sqsubseteq v$. □

So, an element of L_{n+1} is either an element of L_n, or a pair of elements where the first component is either an atom or an element of L_n and the second component is an element of L_n. We write the pair (a, l) by $a{:}l$ using the pairing operator which is assumed to be right associative. For example, $a{:}b{:}c = a{:}(b{:}c)$.

Next we define the projection mapping from L_{n+1} to L_n. This mapping acts as identity on the lists of L_{n+1} that also belong to L_n. For the other lists its value is the list in L_n that best approximates the argument.

Definition

The *projection functions*, $\psi_n : L_{n+1} \to L_n$, are defined inductively:

$$\psi_0(x) = \begin{cases} \Diamond & \text{if } x = \Diamond; \\ \bot & \text{otherwise.} \end{cases}$$

$$\psi_n(x{:}y) = \begin{cases} x{:}y & \text{if } x{:}y \in L_n; \\ x{:}\psi_{n-1}(y) & \text{if } x \in A; \\ \psi_{n-1}(x){:}\psi_{n-1}(y) & \text{otherwise.} \end{cases}$$

□

Examples

L_2 contains $\bot$, $\Diamond$, $1{:}\Diamond$, $(\Diamond){:}\bot$, $1{:}\bot$, $1{:}2{:}\bot$, $(1{:}\bot){:}(\bot){:}\bot$, $\bot_A{:}\bot$, etc.

L_3 contains all the elements of L_2 as well as $1{:}2{:}3{:}\bot, (1{:}2{:}\bot){:}(1{:}\Diamond){:}(\Diamond){:}\bot$, etc.

Examples of the ordering and projection functions:

$$\bot \sqsubseteq \Diamond, \quad \bot_A{:}\bot \sqsubseteq 1{:}\bot \sqsubseteq 1{:}2{:}\bot, \quad (1{:}\bot){:}\Diamond \sqsubseteq (1{:}\Diamond){:}\Diamond,$$

$$\psi_2(1{:}2{:}\Diamond) = 1{:}2{:}\Diamond, \quad \psi_2(\,(1{:}2{:}\bot){:}(1{:}\bot){:}(\bot){:}\bot\,) = (1{:}\bot){:}(\bot){:}\bot.$$

3 Infinite lists

Infinite lists will be constructed to be sequences of finite lists where the nth element comes from L_n. Each element of the sequence will be the ψ-projection of the next. This is the consistency condition necessary for the inverse limit construction.

Definition

The set of ***infinite lists***, L_∞, is the inverse limit of $\{L_n; \psi_n\}$:

$$L_\infty = \{\langle s_n \rangle_{n=0}^{\infty} \mid \text{ for all } n,\ s_n \in L_n \text{ and } s_n = \psi_n(s_{n+1})\}.$$

□

Clearly, there is a canonical injection from each L_n into L_∞ where each element is mapped to an infinite sequence whose first n elements are the iterated projections while the rest are constant injections.

Next, make L_∞ into a partial order by defining the order componentwise on the sequence.

Definition

Let $l_1 = \langle s_n \rangle$ and $l_2 = \langle t_n \rangle$ be infinite lists. The ordering on infinite lists is defined by:

$$l_1 \sqsubseteq l_2 \qquad \text{iff} \qquad s_n \sqsubseteq t_n \text{ for all } n.$$

□

4 Complete Partial Orders

Once infinite lists are given with a partial ordering, the next step is to look at a chain of lists. We want each chain to have a least upper bound that is again an infinite list, i.e., a member of L_∞.

Definitions

A set of lists $\{l_i\}_{i=0}^{\infty}$ is called a ***chain*** if $l_i \sqsubseteq l_{i+1}$ for all i. An element is an ***upper bound*** of a chain if it is larger than every element of the chain. The ***least upper bound*** is an upper bound that is least among all the upper bounds. We denote the least upper bound of a chain $\{l_i\}_{i=0}^{\infty}$ by $\bigsqcup_{i=0}^{\infty} l_i$. A partial order where every chain has a least upper bound is called a *complete partial order.* □

With these definitions we prove the Lemma needed to show that L_∞ is a complete partial order, i.e. that all chains have a least upper bound.

Lemma

Each L_n is a complete partial order.

Proof. We show that the least upper bound exists for every chain by showing that a chain in L_n can have only a finite number of distinct elements, and therefore, the maximum of the chain is the least upper bound.

Let us define the ***rank*** of a list to be the total number of occurrences of the pairing operator (:), the empty list ($\diamond$), and defined elements of A. We claim that: (a) for each L_n the maximum rank is bounded, and (b) if l_1 is ***strictly*** less defined than l_2, then the rank of l_1 is less than the rank of l_2. (a) is proved by induction on the rank of lists. The max of the rank in L_0 is 1. A list in L_n consists of a finite number of lists of lower order. (b) is true because the only way to make a list strictly more defined is by either replacing $\perp$ by $\diamond$ or $x{:}\perp$ for some x, or replacing $\perp_A$ by a

defined element of A, all of which increases the rank by one. Therefore, since the rank is bounded, every chain must be finite. □

Next we prove the main Theorem that L_∞ is a complete partial order by explicitly constructing a list that is the least upper bound of a chain, and showing that it is a member of L_∞.

Theorem

L_∞ is a complete partial order.

Proof. Let $\{l_i\}$ be a chain with $l_i = \langle l_{ij} \rangle_{j=0}^{\infty}$. Construct a new list as follows:

$$l = \langle s_n \rangle_{n=0}^{\infty} \qquad \text{where} \qquad s_n = \bigsqcup_{i=0}^{\infty} l_{in}.$$

From the previous Lemma, the least upper bound, s_n, exists for each n, so l is well defined. By definition s_n is in L_n. To show that l belongs to L_∞, we need to show $s_n = \psi_n(s_{n+1})$ for all n. This is shown by:

$$\begin{aligned} \psi_n(s_{n+1}) &= \psi_n(\bigsqcup_{i=0}^{\infty} l_{i\,n+1}) \\ &= \bigsqcup_{i=0}^{\infty} \psi_n(l_{i\,n+1}) \\ &= \bigsqcup_{i=0}^{\infty} l_{in} \\ &= s_n. \end{aligned}$$

The first, third and last equalities are by definitions. The second equality follows from the monotonicity of ψ_n and the fact that every chain is finite.

So, l is an upper bound since each component is the least upper bound, and it is the least because it is the least element componentwise. □

5 Application to Program Semantics

With L_∞ proved to be a complete partial order, we can use it for the fixed point approach to program semantics. The following programs are defined using an applicative language with lazy evaluation like Turner's *KRC* [82].

Example 1

A program to generate an infinite list of 1's is:

$$f = \tau[f] \quad \text{where} \quad \tau[f] = 1 : f.$$

The meaning of f is the least upper bound of the partial lists defined by the τ_i's:

$$[\![f]\!] = \bigsqcup_{i=0}^{\infty} \tau_i,$$

where the τ_i's are given inductively by:

$$\tau_0 = \bot \quad \text{and} \quad \tau_{n+1} = \tau[\tau_n] = 1 : \tau_n.$$

By considering a finite list τ_i to be both a member of L_i as well as its injection into L_∞, the first few terms are:

$$\begin{aligned}
\tau_0 &= \bot \\
\tau_1 &= \tau[\tau_0] = 1 : \tau_0 = 1 : \bot \\
\tau_2 &= \tau[\tau_1] = 1 : \tau_1 = 1 : 1 : \bot \\
&\vdots \\
\tau_i &= \tau[\tau_{i-1}] = 1 : \tau_{i-1} = 1 : \cdots : 1 : \bot
\end{aligned}$$

and therefore

$$[\![f]\!] = \langle \tau_i \rangle_{i=0}^{\infty}$$

which is the object representing an infinite list of 1's.

Example 2

A program to generate a list of order ω. First, define a program that generates lists of all finite order by using a parameter:

$$\begin{aligned}
f0 &= 1 : f0 \\
f(n+1) &= fn : f(n+1).
\end{aligned}$$

Next we diagonalize to get higher order elements:

$$Fn = \tau[F]n \quad \text{where} \quad \tau[F]n = fn : F(n+1).$$

If $[\![fn]\!] = \sigma^n$ for each n, then

$$\begin{aligned}
\sigma^0 &= 1 : 1 : 1 : \cdots, \\
\sigma^1 &= \sigma^0 : \sigma^0 : \sigma^0 : \cdots, \\
\sigma^2 &= \sigma^1 : \sigma^1 : \sigma^1 : \cdots, \\
&\vdots,
\end{aligned}$$

and

$$[\![F0]\!] = f0 : f1 : f2 : \cdots = \sigma^0 : \sigma^1 : \sigma^2 : \cdots.$$

Though the meaning of $F0$ is precisely given, this is not satisfactory because, in practice, since it is an infinite list, σ^0 cannot be given fully before giving σ^1. What we need is to represent all infinite lists as limits of finite ones. To do this we need projection functions from infinite lists to their finite approximations.

Let $p_n: L_\infty \to L_n$ be the projection function defined by:

$$p_n(\sigma) = \sigma_n \quad \text{where} \quad \sigma = \langle \sigma_n \rangle_{n=0}^{\infty}.$$

Then the meaning of $F0$ can be given as:

$$[\![F0]\!] = \bigsqcup_{i=0}^{\infty} \tau_i 0,$$

where the finite lists are defined inductively by:

$$\tau_0 k = \bot \quad \text{and} \quad \tau_{n+1} k = \tau[\tau_n]k = p_n(\sigma^k) : \tau_n(k+1) \quad \text{for all } k.$$

More explicitly:

$$\begin{aligned}
&\tau_0 0 = \bot \\
&\tau_1 0 = \tau[\tau_0]0 = p_0(\sigma^0) : \tau_0 1 = \sigma_0^0 : \bot = (\bot) : \bot \\
&\tau_2 0 = \tau[\tau_1]0 = p_1(\sigma^0) : \tau_1 1 = p_1(\sigma^0) : p_0(\sigma^1) : \tau_0 2 = \sigma_1^0 : \sigma_0^1 : \bot \\
&\quad \vdots \\
&\tau_n 0 = p_{n-1}(\sigma^0) : p_{n-2}(\sigma^1) : \cdots : p_0(\sigma^{n-1}) : \tau_0 n.
\end{aligned}$$

The projection functions ensure that $\tau_n 0$ is an element of L_n. At each stage, not only does the length of the list grow, but each component of the list becomes more and more defined.

6 Conclusion

In order to construct ω-order infinite lists, we used the inverse limit construction technique from topology. This seems to be a very general technique for going from finite to infinite. Our construction of the L_n's was chosen because of its simplicity and so may be less intuitive than others, although any choice would probably lead to isomorphic limit. The only non-trivial part of these notes is in showing that the least upper bound belongs to the domain of infinite lists we constructed. Here we used the fact that for finite chains a monotonic function can be moved inside the limit.

In de Bakker and Zucker [83] concurrent processes denote infinite trees that are constructed by first defining a distance metric between finite trees, and then, by using the standard completion technique, obtain the infinite ones as limits of Cauchy sequences. By using the inverse limit, we do not need a metric but are able to define the notion of a limit directly.

The inverse limit construction was used by Scott [73] to model the type-free λ-calculus. Infinite lists can be represented as certain terms in the λ-calculus, and therefore, from a theoretical point of view, the standard semantics using D_∞ is sufficient. From a practical and pedagogical point of view, however, a more direct construction using finite lists is desirable, hence this paper.

Future work will be to explore the topological structure of the space of infinite lists, and make the connection with type-free models of λ-calculus more explicit.

Acknowledgments

These ideas crystallized out of the many discussions with Prof. Alain Martin and Chi Fai-Ho. Thanks also go to Jerry Burch for pointing out an error. I would also like to thank Prof. James Kajiya and Lenny Rudin for discussions on the inverse limit construction.

References

J. W. de Bakker and J. I. Zucker,

[83] Processes and the Denotational Semantics of Concurrency, *Foundations of Computer Science IV*, (eds.) J. W. de Bakker and J. van Leewen, Mathematical Centre Tracts 159, pp. 45–100, 1983.

J. Dugundji,

[66] *Topology*, Allyn and Bacon, Boston, 1966.

J. Nagata,

[68] *Modern General Topology*, North-Holland, Amsterdam, 1968.

D. Scott,

[73] Models for various type-free calculi, in *Logic, methodology and philosophy of science IV*, Ed. P. Suppes (North-Holland, Amsterdam, 1973) pp. 157-187.

D. A. Turner,

[82] Recursion equations as a programming language, *Functional Programming and its Applications*, (eds.) J. Darlington, P. Henderson, and D. A. Turner, Cambridge University Press, pp. 1–28, 1982.